Wireless Netw (2012) 18:185–197
DOI 10.1007/s11276-011-0394-z

Converged multimedia services in emerging Web 2.0 session mobility scenarios

Michael Adeyeye · Neco Ventura · Luca Foschini

Published online: 29 October 2011

Abstract The increasing request for converged multimedia services have motivated relevant standardization efforts, such as the Session Initiation Protocol (SIP) to support session control, mobility, and interoperability in all-IP next generation wireless networks. Notwithstanding the central role of SIP in novel converged multimedia, the potential of SIP-based service composition for the development of new classes of Web 2.0 services able to interoperate with existing HTTP-based services is still widely unexplored. The paper proposes an original solution to improve online user experience by integrating a SIP stack into the Web browser, thus enabling the execution of novel SIP-based applications directly at the client endpoint. In particular, our browser extension coordinates with our novel SIP-based Converged Application Server to enable session mobility and prevent abuses of the services available in the client. Experimental results show that our SIP-based solution is feasible in most common Internet deployment scenarios and enables session mobility with limited management cost.

Keywords Communication system signaling · SIP · World Wide Web—Mashups · Browsers · Session transfer

M. Adeyeye
Department of Information Technology, Cape Peninsula University of Technology, Cape Town 8000, South Africa
e-mail: adeyeyem@cput.ac.za

N. Ventura
Department of Electrical Engineering, University of Cape Town, Cape Town, South Africa
e-mail: neco@crg.ee.uct.ac.za

L. Foschini (✉)
DEIS, Università di Bologna, Bologna, Italy
e-mail: luca.foschini@unibo.it

1 Introduction

The convergence of the Internet, broadcasting, and telecommunications has resulted in novel converged multimedia services in the World Wide Web community and paved the way to a new rich research field characterized by several different technological possibilities, service opportunities, and business models, often referred to as Web 2.0. This novel scenario has been enabled by the ever increasing diffusion of new Web-related innovative technologies including new browsers (e.g., Mozilla Firefox and Google Chrome), standardized client-side scripting languages (e.g., JavaScript), original communication models (e.g., Asynchronous JavaScript + XML − AJAX), and novel extensions via browser-specific Application Program Interfaces (APIs). A good example is the Mozilla Community, which has over 1,800,000,000 extensions downloaded for the Mozilla Firefox browser [1].

Despite the great potential stemming from the Web 2.0 service-provisioning scenario, one of the most challenging open issues in supporting converged multimedia services is session management. Broadly speaking, session management is the capacity to dynamically adapt, modify, and maintain active the service session state notwithstanding the several possible changes in the service delivery environment, such as: user mobility between different wireless access points; possible quality level variations due to connection technology change; and executing environment modifications due to application or access terminal change, defined as session mobility. By focusing on session mobility for Web 2.0, various research works in the last decade have explored HTTP session mobility among different hosts and browsers [2–4]. However, despite their potential, none of the above HTTP session mobility solutions has penetrated into the mobile multimedia market,

primarily because of the lack of widely accepted session control standards and consequently due to the limited possibilities to integrate those solutions into fully-converged Web 2.0 services melting together Internet and telecommunications worlds.

The paper tackles the above issues by proposing a novel solution with three original core contributions. First, it enables open and interoperable session management at the client-side (Web browser) by adopting an application-layer approach based on the Session Initiation Protocol (SIP) and SIP overlay infrastructure for session control in next generation fully-converged wireless networks. Second, it proposes a novel SIP-based HTTP session mobility approach that enhances the Web 2.0 user experience and prevents possible abuses of the services available through the introduction of our novel Converged Application Server (CAS) to control HTTP session mobility between browsers. Third, it shows how our enhanced SIP-enabled Web browser not only supports session mobility, but also enables the creation of new Web 2.0 services and service mashups. Our work is fully compliant and seamlessly integrates with standard and widely diffused technologies. The proposed HTTP session mobility extension, called TransferHTTP, has been implemented as a SIP-stack-based extension that leverages the Open Source Package Mozilla Firefox. In addition, the Mobicents Communications Platform [5], an open platform that aims to drive convergence, has been adopted to realize CAS component for session control. The proposed components, part of our wider project on SIP-based session management, are publicly available for wireless practitioners and the SIP community[1] [6–8].

The paper structure is as follows. First, we give the needed background about SIP and then we present the application scenario and the main design guidelines of our distributed architecture. Then, we details our SIP-based Web session blocking/forwarding management solution and protocols, while we give implementation details and some seminal performance results about TransferHTTP extension and CAS. Afterwards, we present the related work, by especially focusing on the very recent Google Wave project, and we discuss our API. Finally, remarks and future work directions end the paper.

2 SIP background

SIP is an application protocol that allows services to participate in session initiation and management. SIP supports different kinds of mobility, namely session mobility, personal mobility, terminal mobility and service mobility [9]. This section briefly introduces the needed background material about SIP.

SIP allows the creation, modification, and termination of service sessions independently of the underlying data-link layer technologies and transport protocols. SIP has been widely applied to voice and video call/conference services over the traditional Internet. Recently, SIP has gained widespread acceptance also in the mobile world as the control protocol for converged communications over all-IP networks (see the third Generation Partnership Project—3GPP—and the IP Multimedia Subsystem—IMS—[10]).

The SIP infrastructure is highly open and flexible, and offers facilities to service developers. In fact, SIP not only defines protocols and messages for session signaling, but also proposes a wider framework, e.g., for decentralized proxy-based session management, endpoint localization, and presence detection. The core entities of the SIP infrastructure are user agents (UAs), registration and location servers, proxies, back-to-back user agents (B2BUAs), and re-direct servers. A UA is a SIP endpoint that controls session setup and media transfer; each UA is identified by a unique HTTP-like Uniform Resource Identifier (URI), e.g., sip:user@domain. The registration server is a naming service that receives register requests by UAs and is able to resolve current UA locations; it interacts with the location server to store correspondences between UA SIP URIs and their current endpoints. Each session or dialog is setup between two UAs, and SIP distinguishes two roles: the requesting UA Client (UAC) and the target UA Server (UAS). Re-direct servers, proxies, and B2BUA contribute to locate SIP endpoints and to route SIP messages. Re-direct servers and proxies represent the core SIP routing infrastructure, but they have limited possibilities to change ongoing dialogs and SIP messages (they can neither generate new SIP requests nor change message content, e.g., modifying media endpoints). B2BUAs, instead, are logical entities with both UAC and UAS capabilities that have full control over traversing dialogs and SIP messages. Consequently, SIP proxies only participate to message routing (and not to media delivery), while B2BUAs, given their UA capabilities and their ability to change SIP messages content, can potentially participate also to multimedia content transport/adaptation by splitting client-to-server direct media paths.

Finally, SIP defines a set of protocols and messages, such as INVITE, REGISTER, REFER, MESSAGE, OK, and ACK, to control sessions; SIP makes no assumption on transport layer, but usually employs UDP. More details about SIP protocol and messages can be found in [10]. Moreover, SIP can be easily extended to support session mobility; for instance, UAC can use the REFER message to redirect the session control flow and specific MESSAGE payloads to move ongoing session state.

[1] Additional information and the prototype code of our TransferHTTP extension are available at: http://www.transferhttp.mozdev.org.

3 Application scenario and distributed architecture

When surfing the Web, the two prominent ways of sharing information or asking a friend to view the same Webpage a referrer is viewing are, currently: using a third party software (Instant Messengers, such as GTALK and Yahoo Messenger) and sending the URL in an email. However, both those alternatives are error-prone, non-automated, and time-consuming for the final users. Therefore, session mobility has been widely recognized in the last years as a core facility to enable new Web 2.0 browsing information sharing experiences [6, 7].

In this paper, we define session mobility as the facility that enables automatic movement of existing Web sessions between two browsers, typically running on different hosts. We distinguish two main types of session mobility: content sharing and session transfer. Content sharing is defined as the ability to simultaneously view the same Web page on two browsers at the same time, by typically transferring only the Web page Universal Resource Locator (URL). Session transfer is defined as the ability to move the whole Web session, including not only the Web page URL but also all needed session data (cookies, session tokens, and visited URLs). An example of content sharing is Alice referring Bob to visit the same news Website that she is browsing: in this case, she would only want to send the URL. An example of session transfer, instead, is moving a Web email session between two Personal Computers (PCs) with the final goal of continuing to check emails, with the same view of read/unread/cancelled messages, etc., without having to sign in again.

Since content sharing and session transfer are critical operations, potentially prone to security problems such as malicious users acting as men-in-the-middle between two interacting parties or possible abuses of services offered by the browsers, we introduced two facilities—Web session blocking and Web session forwarding—at the proxy of the system to control the interaction between the browsers. These are standard facilities in telecommunications, where phone calls can be blocked, forwarded or screened. A facility or service can be defined as a value-added functionality provided by a network to its users [11]. Although call blocking, call forwarding, and similar facilities are specific to telecommunications, they are feasible in the Web-browsing context owing to the interactions between two or more browsers.

To prevent potential security threats typical of end-to-end direct interactions (such as direct end-to-end session redirection based on standard SIP REFER message only), we claim the relevance of proxy-based solutions, where a trusted proxy entity is interposed among the two interacting browsers to mediate and grant their interactions. Following this main design guideline, we propose the distributed architecture shown in Fig. 1; we define our architecture as hybrid-based because it is based both on an infrastructure (proxy) and a client-side parts and it integrates different protocols (namely SIP and HTTP).

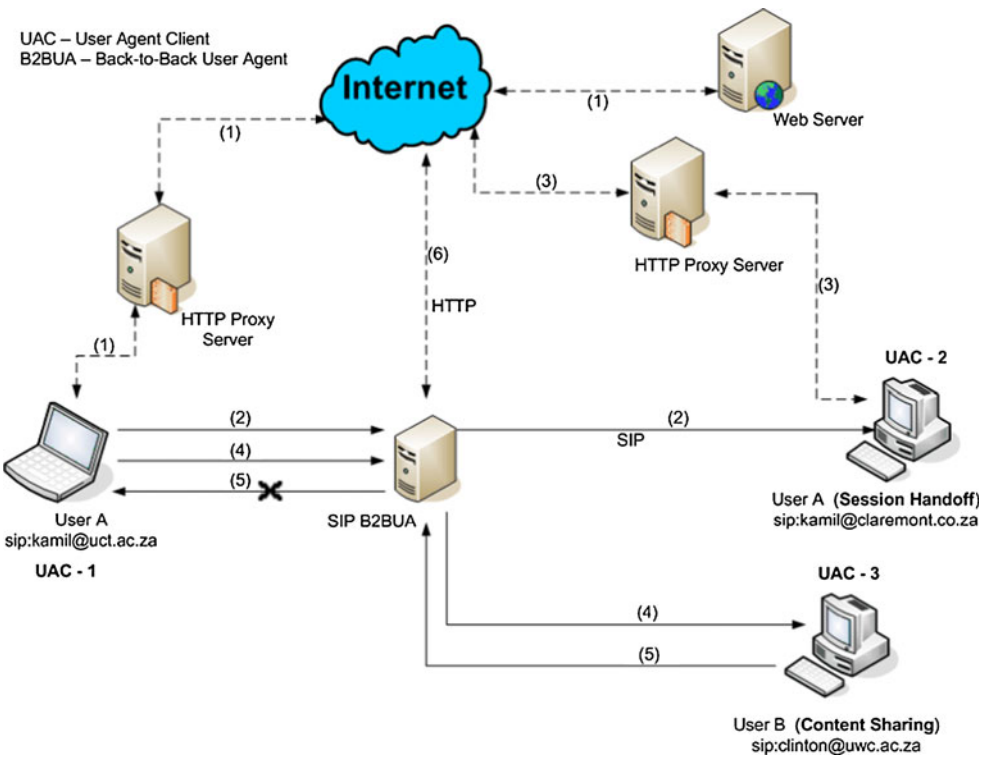

Fig. 1 The hybrid-based architecture with all main interactions

The interaction is between browsers, enhanced with SIP capabilities and acting as SIP UACs, and a SIP B2BUA (proxy) that coordinates browser-to-browser interactions and enforces Web session blocking or forwarding (via SIP, continuous arrows in Fig. 1). Once session state has been moved to the destination browser, it can rebind the ongoing session by interacting with the local HTTP proxy server (via HTTP, dashed arrows in Fig. 1); Fig. 1 reports three significant use cases. In the first one, User A (UAC-1 with SIP URI sip:kamil@uct.ac.za) who is at work, transfers his Web session to his browser (UAC-2 with SIP URI sip:kamil@claremont.co.za) running on his desktop at home (interactions 2–3 in Fig. 1), while in the second one, User A refers User B (UAC-3 with SIP URI sip:clinton@uwc.ac.za) to the same Web page he is viewing (interaction 4 in Fig. 1). In the above two use cases, the proxy allows the required session mobility request by forwarding it. In the third one, instead, User A has set the SIP B2BUA to block all requests from User B (interaction 5 in Fig. 1). In this case, our fine grained session control permits to define and enforce asymmetric session blocking/forwarding: User A can refer User B to view the same Web page, but User B cannot refer User A. Finally, as better explained in Sect. 5.2, the proxy provides a set of Web pages for access control policies configuration (via HTTP, interaction 6 in Fig. 1).

4 SIP-based web session blocking/forwarding management

In order to enable session mobility (either content sharing or session transfer), we propose to extend the Web browser by adding our new Web session transfer/receipt functions. Figure 2 shows the data flow diagrams—DFD—of session mobility transfer/receipt, respectively, at origin (Fig. 2a) and destination (Fig. 2b) browsers. We only require users to enter their information, such as SIP username and password used by UAC to register into the SIP infrastructure, and the destination SIP B2BUA address (see the two arrows entering the browser from the left side in Fig. 2a). The SIP B2BUA address is configured once, at configuration time; the user information, instead, are required from the user at browser activation time and can be saved among user passwords.

The Web session transfer/receipt functions are implemented by our SIP UAC browser extension, called TransferHTTP, that interacts with our new SIP B2BUA, called Converged Application Server (CAS), to manage session state blocking/forwarding. The origin browser (see Fig. 2a) interacts locally with TransferHTTP by passing to it the SIP user information and the destination SIP address, once at activation time, and then ongoing session data and URLs for each required session transfer. TransferHTTP processes session transfer requests and emits SIP MESSAGE with session information to CAS. Afterwards, CAS checks the session mobility request against destination user's accessibility policies, and then forwards the SIP MESSAGE to TransferHTTP extension at the destination browser that, in its turn, generates a local notification to the browser (see Fig. 2b). If the notification is accepted, the session data will be forwarded to the browser and the session will be re-established by pulling needed resources from the Internet.

Figure 3 shows the proposed SIP-based Web session blocking and forwarding protocols and message exchanges. After an initial registration phase (steps 2–3 and 5–6 in Fig. 3) during which TransferHTTP components (SIP UACs) register to CAS (acting also as SIP location/registration server in our infrastructure); then, subsequent SIP message exchanges transverse CAS (SIP B2BUA). For each content sharing or session state transfer request, realized both as specific SIP MESSAGEs encapsulating all needed session state information, CAS enforces access policies by either blocking or forwarding the request towards destination TransferHTTP (steps 7–9). Then, for content sharing, the HTTP Request/Response will be between the Web server and UAC 1 (step 12a) as well as the Web server and UAC 2 (step 12b). In session transfer, instead, the HTTP Request/Response will only be between the Web server and UAC 2 (only step 12b). Finally, when users decide to close their browsers, SIP UACs de-register (steps 13–16).

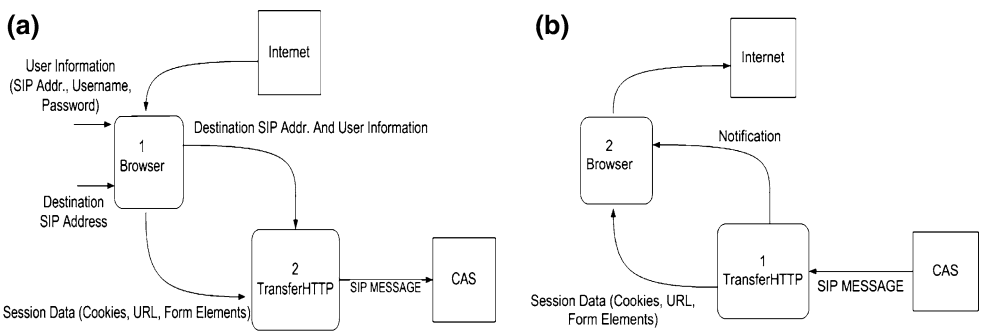

Fig. 2 Web session transfer/receipt at origin (**a**) and destination (**b**) browsers

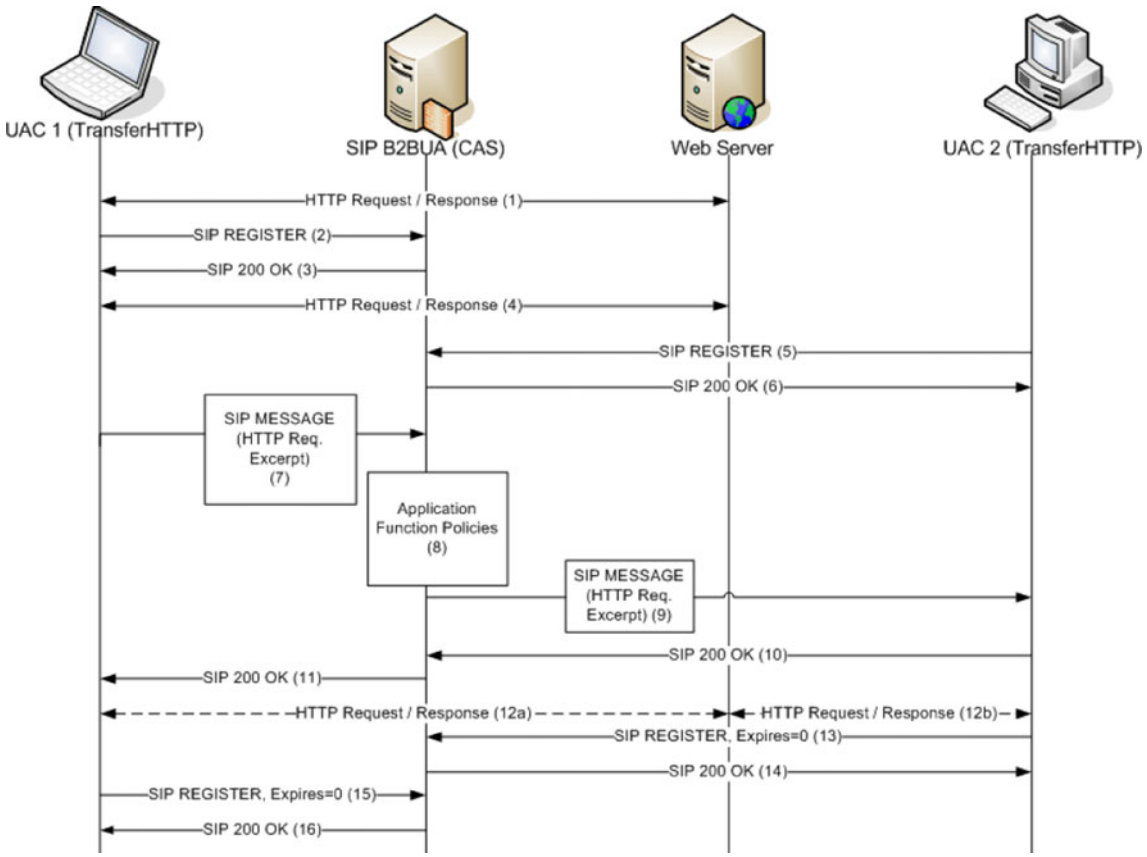

Fig. 3 The message exchange between Web clients

By focusing on exchanged SIP messages, the session data of a Web session transfer request is sent in an XML format using the SIP MESSAGE method [7], and it could consist of a URL, cookies and session tokens depending on the kind of request. As regards session blocking, when CAS detects an unauthorized session mobility request, it responds to the SIP MESSAGE with a "Forbidden (403)" response, notified to the user as a warning message in browser status bar.

To conclude, let us stress that the availability of a SIP stack integrated within the client browser really enables new converged Web 2.0 service provisioning scenarios. For instance, our TrasferHTTP extension not only enables session mobility, but also permits to initiate SIP-based Voice over IP (VoIP). Moreover, with our extension, browsers can take different SIP roles and act not only as UAC, but also as UASs able to deliver SIP-based services required by other external SIP UACs.

Therefore, we strongly believe that our TransferHTTP, together with the availability of new media processing and playing capabilities in next generation Web browsers, makes possible new innovative SIP-based service mashup scenarios. For instance, the latest version of TransferHTTP supports a highly dynamic multimedia stream sharing service, called "Stream Media to Call", that permits users to use their Web browser as a streaming server, thus making available their multimedia contents as video/audio streams deliverable to authorized buddies.

5 Implementation and performance results

This section first briefly sketches some implementation details about our novel TransferHTTP extension and CAS SIP B2BUA component; and then, it presents experimental results aimed to evaluate the cost of the proposed session mobility management solution.

5.1 TransferHTTP browser extension

TransferHTTP has been realized as an extension for Mozilla Firefox Web browser. The services available in the TransferHTTP extension include not only content sharing and session transfer, originally presented in this paper, but also VoIP ("Make a call"), and multimedia content sharing ("Stream Media to Call").

Our session mobility facility consists of two main parts: a SIP stack and our TransferHTTP extension. The SIP

stack used in this implementation is the PJSIP [12], which is an Open Source project and small footprint multimedia communication libraries written in C. We created a shared library (1.7 MB in size) from the SIP stack and the shared library interacts with the TransferHTTP extension via a Cross Platform Component Object Model (XPCOM). In particular, we added our extension as a new XPCOM with the contract id "@ngportal.com/SIPStack/SIPStackInit;1" into the existing XPCOMs in the browser [13].

As regards our TransferHTTP extension, it has been realized atop built-in interfaces in Mozilla Firefox version 2.0–3.5 that include nsIPref, nsICookie/nsICookieManager/nsICookieManager2, nsIThread, nsIPasswordManager/nsILoginManager, nsIIOService, nsIPromptService, nsITimer, nsIObserver; in addition, it makes available a new interface, named "ImyStack" to exposes TransferHTTP functions. The underlying implementation, written in C++, is scriptable and developers can use JavaScript to invoke its functions. In particular, TransferHTTP API exposes signaling functions, typically available in telecommunications frameworks, to create innovative applications at the client side. For instance, our API makes it possible to create applications that can use the instant messaging and presence features in SIP [14]. The TransferHTTP client API currently exposes the SIP REGISTER method so that a browser can register with a SIP network. It also exposes the SIP MESSAGE method to send messages or chat and the SIP INVITE method to make calls between two browsers. Using the TransferHTTP API, it was easy to implement the HTTP session mobility service presented in this paper. Other services, such as Stream Media to Call to broadcast media are also based on the same APIs.

Our extension also offers a graphical user interface, shown in Fig. 4. In particular, TransferHTTP adds a new menu to the menu bar in the browser and a "Preferences" submenu to let users edit configuration parameters. The settings include the SIP proxy address/port number and the SIP username. Figure 4 shows the new menu and all available options in the status bar. The "Make a call" option enables a user to make a VoIP call to another, and the "Stream Media to Call" one enables a user to stream media to another user. Other options, namely "Register Client", "De-register Client", and "Accept Session," are used to register to a SIP Registrar, de-register the client and accept a Web session transfer request sent to the browser, once controlled and forwarded by CAS. To enjoy the full potential of TransferHTTP, developers are advised to

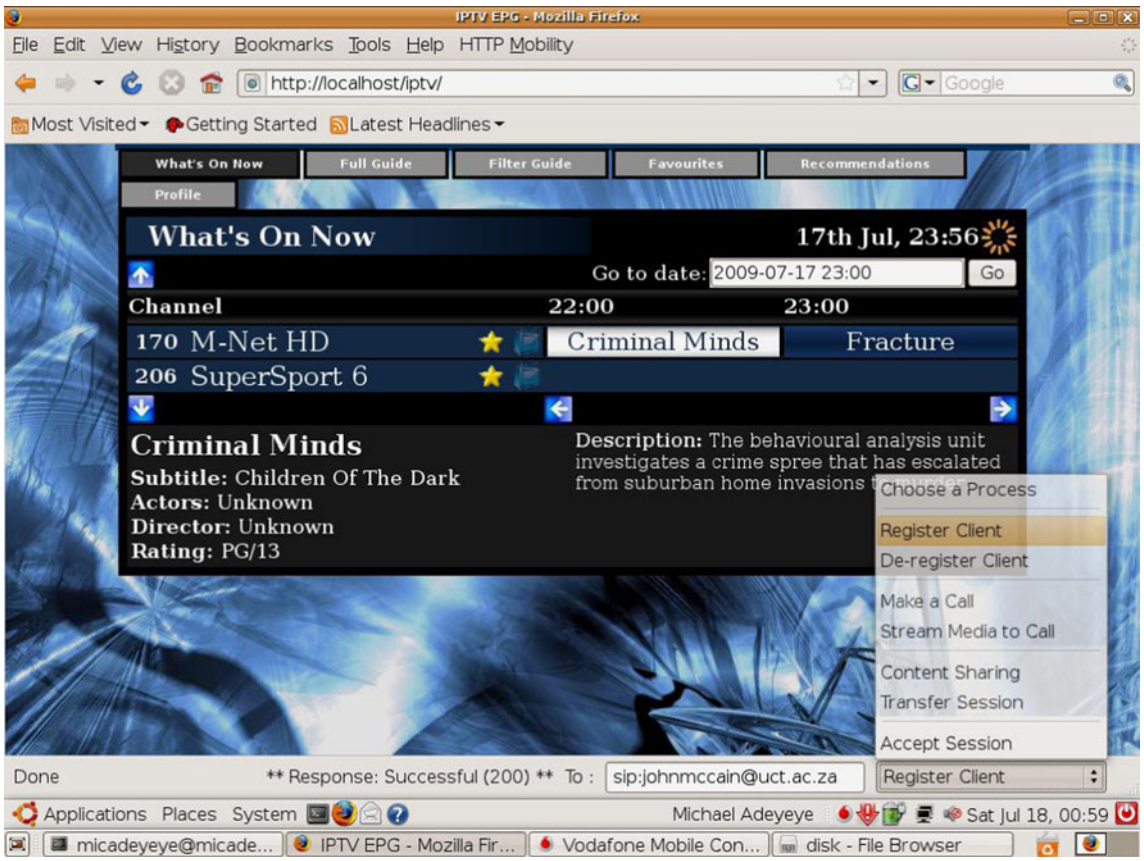

Fig. 4 The client features and user interface

develop XML User Interface Language (XUL)-based applications. XUL was the language used to develop the Mozilla Firefox user interface. Hence, a XUL-based application developed by any interested user will be able to use the APIs in the Web browser and our extension.

Various versions of TransferHTTP extension are available. Version 1.2 was meant for Peer-to-Peer interaction, version 1.3 worked in a client–server environment, and current version 1.4 supports also VoIP and our novel the Stream Media to Call service. The browser extension, developed for the Mozilla Firefox open source browser, is publicly available under the Mozilla Public License (MPL). We hope Web 2.0 and SIP research practitioners will continue to extend the API and creating more innovative user-generated services [13].

5.2 CAS SIP B2BUA component

The Mobicents Communications Platform was used to implement CAS [5]. Mobicents is an open Java-based platform that enables creation, deployment and management of services and applications that integrate voice, video and data across a range of Internet Protocol (IP) and communications network by multiple devices. It implements and delivers both competing and interoperable programming models—Java APIs for integrated networks (JAIN) service logic execution environment (SLEE), and SIP servlets—to develop Web and VoIP applications that work together.

CAS implements Web session blocking and forwarding. Although applications could be developed using either JAIN SLEE or SIP servlets in the Mobicents Platform, the current implementation is based on the Mobicents SIP servlets programming model [15]. When the proxy receives a SIP request, the application router (AR), a standard component part of the SIP servlets model, is called by the Mobicents container. When a request addressed to CAS arrives, AR selects our SIP servlet application—implementing CAS session control logic for incoming SIP REGISTER and MESSAGE requests—to process it.

CAS offers also a Web user interface (not shown here) with three main pages. The "Session Tracking and Pickup" page logs all session transfer requests, call setup requests, and actions taken on them. For each session transfer, it provides the source SIP address, the destination SIP address, the SIP method, date, action taken (also known as status) and the referred URL in the case of a session transfer request. The "My Account" page enables a user to set access policies; information available here includes SIP URIs with their log-in details and policies. Using this page, it is possible to instruct CAS about how to handle content sharing and session transfer requests. Finally, the "Buddy List" page contains a list of his contacts; users can add new contacts and check their Buddy List page to see if their contacts are online or offline.

5.3 Performance results

We have thoroughly tested and evaluated the performance of our SIP-based session mobility solution by deploying it in a real-world testbed: TransferHTTP extension runs at several Linux client laptops equipped with Mozilla Firefox and the shared library developed from PJSIP [12], while CAS SIP servlets execute on a standard Windows XP box with 3 GHz Pentium 4 processors and 1,024 MB RAM. As outlined in Sect. 5.1, the integration of the SIP stack into the Mozilla Firefox browser was achieved in a loosely-coupled approach via XPCOM [13].

Reported experimental results focus on session blocking/forwarding functions needed to enable both our content sharing and session transfer use cases. The reported evaluation is aimed both to confirm the feasibility of the proposed approach and to inspect the effectiveness and the efficiency of CAS, that is the core and the most loaded session control component, and of TransferHTTP, that executes at the client side and thus has to be also carefully assessed. We have collected four main performance results: (1) preliminary results about additional delay due to CAS interposition in the browser-to-browser path; (2) CAS memory consumption and processing delay under different load conditions; (3) CAS performances under system overload; (4) TransferHTTP memory consumption at the client side.

The first set of experiments was aimed to preliminarily assess the latency introduced by CAS. We measured it for both send/forward and block/forbid requests and we focused on the delay of MESSAGE requests without considering TransferHTTP registration time because the registration phase occurs only once (at user login) and does not affect user-perceived delays at runtime. In addition, we repeated our experiment under two different deployments: in the local deployment, the two clients (browsers) and CAS have been all deployed within a local Intranet (a 100Mbps Ethernet); in the Internet deployment, we deployed the two clients at the University of Bologna (Italy) and the CAS at the University of Cape Town (South Africa). The results for this first test have been collected far from overload conditions. In the local deployment, the latency introduced by CAS to block/forbid the request is always less than one second (approx. 0.017 s), while it takes longer time for CAS to process a send/forward request (approx. 0.128 s). In the Internet deployment, instead, we measured a delay of about 0.512 s to block/forbid a request and 0.655 s to sent/forward a request due to both increased network latency and relatively frequent SIP request re-transmissions caused by packet losses along

Table 1 CAS performance under normal load conditions

No of requests	No of forbidden responses	No of timeout responses	Total responses	Memory consumption (KB)	CPU usage (%)
20	20 (0.58)	0 (0)	20 (0)	1,036 (13.05)	21 (0.58)
40	32 (0.58)	8 (2.08)	40 (0)	875 (12.09)	21 (1)
60	59 (1)	1 (1)	60 (0)	900 (15.82)	21 (0.58)
100	98 (0.58)	2 (0.58)	100 (0)	580 (12.58)	24 (2.08)
120	120 (0.58)	0 (0.58)	120 (0)	700 (13.23)	21 (2.02)
150	149 (0.58)	1 (0.58)	150 (0)	500 (7.64)	24 (1)
200	198 (1)	2 (1)	200 (0)	1,100 (28.34)	24 (1.73)
250	246 (2.08)	4 (1)	250 (0)	448 (9.87)	24 (0.58)
300	297 (2.02)	3 (1.15)	300 (0)	1,200 (13.10)	24 (2.33)
400	397 (2.52)	3 (1)	400 (0)	520 (10)	21 (1)
500	494 (2.33)	6 (2)	500 (0)	1,150 (11.45)	24(1.15)
600	598 (3.05)	2 (0.58)	600 (0)	1,006 (19.78)	24 (1)

No. of requests, fulfilled/timeout/total responses, memory, and CPU usage

the path. All above measurements have exhibited a limited variance, under 5% for more than one hundred runs. However, the performance of the system could significantly improve with a faster Internet connection, especially given the relatively bad conditions of the network trunks throughout Africa.

The second set of experiments focuses on CAS during normal load conditions. Table 1 reports number of forbidden and timed-out responses, total number of responses (forbidden plus timed-out responses), memory utilization, CPU usage, and average time of a request, for different incremental loads, going from 20 messages to 600 messages sent at a constant rate of 1 message-per-second (mps). To collect these performance results we used the SIPp traffic generator, by focusing mainly on block/forbid requests because, in a real deployment, we expect they are the most numerous ones [16]. We repeated the experiment several times for each number of requests: in Table 1 we report mean values and, for each value, its variance in brackets; as reported, all results show a limited variance. Starting from CPU usage, CAS execution imposes a limited CPU load, always below 25% even for the highest number of requests (600 requests). As for the memory, it also showed good performances; at the constant rate, the memory needed to fulfill the increasing number of requests is always below 1.2 MB.

As for forbidden and timed-out responses, also for sake of comparison with our third set of experiments, we plot the number of (correctly) fulfilled forbidden responses (dashed-and-dotted line and y-axis on the left in Fig. 5) and the number of timed-out responses (solid line and y-axis on the right) against the total number of requests (see Fig. 5). Forbidden responses graph is linear and that demonstrates that CAS is able to correctly fulfill (almost) all received

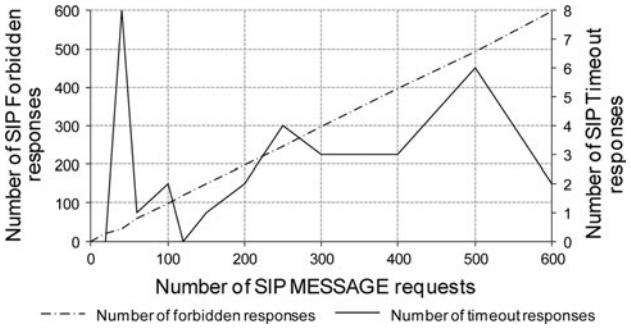

Fig. 5 CAS request versus response performance graph under normal load conditions

block/forbid requests, while the few time-outs that sporadically occur are independent of the number of requests (see Fig. 5).

Finally, let us stress that the results presented here are based on the technical specifications of the Windows XP boxes CAS was deployed on; in particular, to demonstrate the wide feasibility of our solution, we tested CAS in a standard deployment scenario with not-so-powerful hardware and by using standard and non-optimized configurations both for host background services and for the Mobicents server. We observed that the total time required, memory consumption, and CPU usage significantly dropped when CAS was deployed and tested on a high performance PC (a Core 2 Duo PC).

The third set of experiments points out the CAS scalability and correctness under heavy-load condition. To stress CAS, in each test run we generated and sent to CAS several non-authorized session mobility requests to be blocked. In particular, we heavily loaded CAS by using the pjsip-perf SIP traffic generator as the performance measurement tool. Pjsip-perf executed at the client host, sends bursts of SIP

MESSAGE messages, going from 20 to 600 messages-per-burst, at the very high and challenging frequency rate of 677 mps [17].

The resulting average response time of CAS varied from 10,558 to 48,557 ms, and the average number of SIP response messages was more than the average number of SIP requests. For example, when 250 SIP MESSAGE requests were sent, the number of SIP responses was 276. The reason why the number of responses is higher than the number of requests is that SIP supports multiple responses to a request. The responses were SIP 403 Forbidden and SIP 408 Request Timeout (for messages that timed-out at CAS due to the delay caused by excessive overload).

Figure 6 shows CAS behavior under heavy-load conditions (the same type of graph shown in Fig. 5 for normal load conditions with the only difference that there is one only y-axis on the left). We observed that CAS optimum performance is reached at around 100 SIP MESSAGE requests: at this point, CAS can still generate equal number of correct SIP responses (SIP 403 Forbidden). The number of SIP 408 Timeout gradually increases as the number of requests increased, starting from 99 SIP MESSAGE requests, thus showing how CAS performance degrades under system overload. Test results also show that the server could not block all requests: CAS was able to respond to an average number of 73 requests (out of the 600 requests) in less than the response timeout set at the client (which was 32 s), while an average of 531 requests expired at the client due to timeout (see Fig. 6). In any case, requests were queued at CAS that generated responses later on as seen from our test logs; that let us exclude memory (Mobicents SIP message queue length) as a possible CAS bottleneck. Hence, we believe that CAS bottleneck is the number of Java threads devoted to CAS servlet execution in Mobicents.

To completely assess the CAS component under system overload conditions, we also collected memory consumption. As Fig. 7 shows, CAS memory consumption gradually increased as the number of SIP MESSAGE requests in

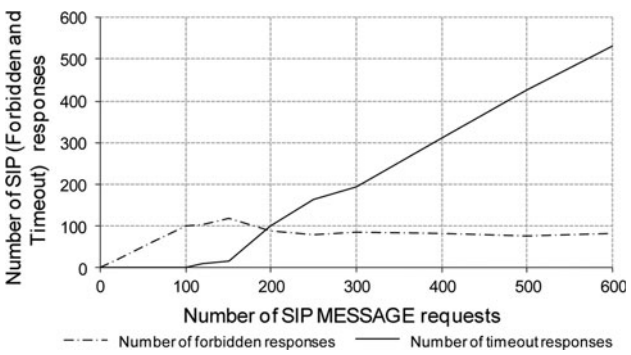

Fig. 6 CAS request versus response performance graph under overload conditions

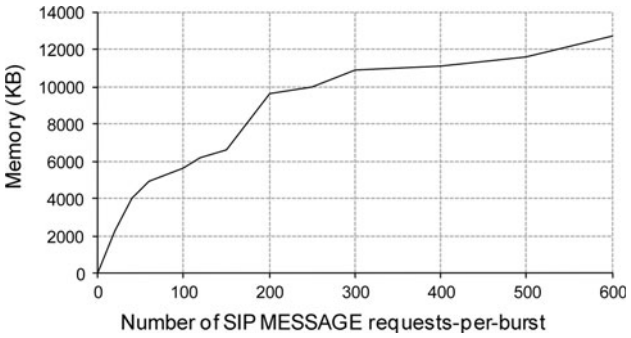

Fig. 7 CAS memory consumption graph

the burst increases. The reason is that the number of threads required to process the requests increases; hence, there is an increment in the memory required to process the requests. In particular, our experiment assessed CAS memory usage under different working conditions. First of all, we experimentally verified that our CAS component—which has been realized as a Java-based Mobicents SIP servlet application—can occupy two main runtime execution states that we call idle and non-idle. In order to reduce the memory consumption on the PC, CAS goes into the idle state whenever, for a certain time period, there is no SIP message to respond to; in that case, it takes some time for CAS to respond to every first subsequent request (due to servlet re-activation).

In idle state, CAS memory usage, collected by using the system performance monitor (Windows Task Manager), is approximately 206 MB out of the 1 GB RAM in the Windows box. In the non-idle state, CAS memory consumption increased by approximately 2.12 MB when a burst of 20 SIP MESSAGE messages was sent to it and by 13 MB when a burst of 600 SIP MESSAGE messages was sent to it (see Fig. 7), while at its optimum performance, around 100 SIP MESSAGE requests, its memory consumption increase is 5.5 MB. Finally, we observed that CAS CPU usage was always at 100% during the entire system overload test.

Our fourth (and last) set of experiments evaluated the cost of running our lightweight TransferHTTP extension at the client side. We used the standard system monitoring tool, available on Ubuntu operating system, to view current processes and to gather the memory consumption data. During the tests, the browser was used to browse the Internet and to move ongoing session (by using both content sharing and session transfer facilities); VoIP and Stream Media to Call services have also been invoked during the test runs. Table 2 shows the browser's memory consumption test.

The SIP stack execution as a background service caused a very contained and low browser's memory consumption increase (TransferHTTP overhead) of 5.83 MB on average.

Table 2 Web browser memory consumption test

	Beginning	2 h	4 h
Without SIP stack integration	11.3	11.6	11.9
With SIP stack integration	16.7	17	18.6

Memory consumption expressed in MB

Findings showed also that the browser neither crashed nor froze, even when running the SIP stack as a background service for very long-run tests up to 4 h.

6 Related work and discussion

This section overviews main related work and ongoing efforts on session mobility and compares them, most especially the recent Google Wave proposal with our solution. In addition, we discuss the benefits of our approach by summarizing our main technical contributions.

6.1 Related work

HTTP session mobility has been carried out using different schemes, namely client-based, server-based and proxy-based architectural schemes [2–4]. A client-based architectural scheme requires modifying the user agent client (UAC) and a server-based architectural scheme requires modifying the user agent server (UAS). UAC and UAS, here, refer to a Web browser and Web server, respectively. In some cases, a Proxy can be placed along the communication path of the client and the server. This is called a proxy-based architectural scheme. While the proxy-based architectural scheme [2] violates the HTTP 1.1 security during its implementation, the client-based architectural scheme [4] requires a repository server, which remains idle when the web client is in use. That is, it does not act as a web cache or a session tracker. In addition, it can only work when a web session has not expired, and the use of a repository server does not make it an intrinsic client-based architectural scheme.

However, it is worth mentioning that most research on mobility focuses on terminal or host mobility. Examples include Jan et al. [18], Snoeren et al. [19], Atiquzzaman et al. [20, 21] and Stewart et al. [22]. In terms of personal mobility, approaches that have been used include Di Stefano et al. [23], Roussopoulos et al. [24], Appenzeller et al. [25], Liscano et al. [26], Herman et al. [27], Raatikainen [28] and Bagrodia et al. [29]. These projects address problems related to Mobile IP and using middleware for mobile networks, to mention a few.

Another area that is currently explored is Media Independent Handover (MIH) between heterogeneous networks. The IEEE 802.21 working group is designing a framework that allows mobile terminals to get information about nearby networks before performing a handover. Some of the works that have explored MIH are Silvana and Schulzrinne [30], Lampropoulous and Passas [31] and Tsagkaropoulos et al. [32]. What distinguishes our proposal from those mentioned above is that it addresses HTTP session mobility, while those mentioned above address terminal mobility. In addition, our proposal addresses HTTP session mobility at the application layer, while those mentioned above address terminal mobility not only in the application layer, but also other OSI layers.

Projects that took advantage of SIP extensibility include work by Shacham et al. [33] and the Akogrimo project [34]. While Shacham et al.'s work [33] extensively explores session mobility issues, also related to the use of the SIP REFER message, the Akogrimo project [34] involves embedding Web service data in a session description protocol (SDP). Although the REFER method is a standardized mechanism to redirect the ongoing session between an old and a new session SIP endpoint, its use alone does not cover the more complex content sharing and session transfer problems. To tackle these issues, in an expired IETF Internet draft [35], two approaches were identified for transferring URL between two Web browsers. The first approach was by sending the URL via a SIP MESSAGE method: this approach is what this work is based on. The second approach was by using SIP NOTIFY method. That approach was described as a way of achieving conference model of Web browsing, whereby a browser could be notified when a Web page, on another browser, has changed. Although the approaches were not standardized, they showed different ways of achieving Web share using SIP MESSAGE and NOTIFY methods.

Another project that exploits the SIP extensibility and very similar to this research was carried out by Munkongpitakhun et al. [36]. In the project, a SIP stack is also integrated into a Web browser, and a SIP MESSAGE method is used to transfer session data as well. Although it is very similar to our proposal, the project, like other related work on HTTP session mobility, only addresses session transfer and not content sharing. In addition, in terms of the signaling, two web browsers have to establish a call session using a SIP INVITE method before a session handoff can take place. Hence, that approach introduces unnecessary overheads in the signaling and it is not clear if users need to be involved in a multimedia session, such as voice call, before session handoff can take place. Finally, implementation details about [36] are limited and no software is available to confirm their findings.

6.2 Discussing emerging web 2.0 models and tools: parlay, Google wave, and TransferHTTP

The need for Open APIs is greatly increasing [14, 37, 38]. The APIs are needed for user-generated services. Although there are APIs, such as Google APIs and Parlay-X APIs, for developing Web 2.0 applications and basic Web service APIs for access to circuit-switched, packet-switched, and IMS networks, they fall short of enabling innovative converged applications or services from users.

Open standard APIs are desirable for introducing new services because they make the separation between the application and the platform explicit. They allow application portability and allow the functions of the platform to be used by multiple applications easily. APIs are application-centric, while protocols are network-centric. APIs allow programmers to focus on the logical flow of applications using only the necessary functions provided by the platform, rather than concerning themselves with low-level details of messages that must flow across the network. As a result, a well-defined API allows the application programmer to work at a higher level of abstraction than that of the protocol [39].

The SIP API reflects the SIP protocol fairly closely [40, 41]. It is useful for situations where the application is rather simple and where the underlying network is an IP network. However, the SIP API is at a lower level of abstraction than the call control APIs, such as JAIN and Parlay APIs. As a result, it offers the programmer finer grained control and better performance than call control APIs.

Another open messaging and presence protocol that is widely used is extensible messaging and presence protocol (XMPP) [42]. Its APIs have been used to develop XMPP clients such as the Google Talk and Pidgin. It is gaining wide acceptance in the software industry, where it is being used to develop communication and collaborative tools. Examples of shared applications built with XMPP are shared whiteboard and chessboard [43]. Another work that is currently exploiting XMPP is the Google Wave.

The Google Wave project currently looks promising for application developers but it is still limited in functionality. The Parlay-X APIs are already claimed to have very limited functionality [37]. The reason is that they are not designed to handle the data model for the entire service or signaling in telecommunications.

Table 3 shows the comparison of our work with the Google Wave. Google Wave uses the open XMPP protocol, so anyone can build their own Wave system [44, 45]. The Google Wave API allows developers to use and enhance Google Wave through two primary types of development, namely Extensions and Embed. The Extensions represent the server side, while the Embed represents the client side. The extensions (also called the Robots API) can be developed using the Java Client Library, Python Client Library, or Gadgets API, while the embed, which is embedded into a Web application, is always written in JavaScript. The Google Wave and TransferHTTP provide the same services, though over different architectures. While Google Wave API is used to develop applications that reside on a Web server, TransferHTTP APIs are used to develop applications that reside at the client end. For example, the Click-to-dial in Google Wave [46] requires the server to set up a call session, while in TransferHTTP, the client sets up the call session. In Google Wave, the robot in the Web server is responsible for initiating the signaling, while in TransferHTTP, the SIP stack in the browser executes the signaling directly.

Table 3 Comparison of Google wave and TransferHTTP + CAS

	Google wave	TransferHTTP + CAS
Technologies		
Client	HTML and JavaScript	XUL and JavaScript
Server	Python/Java/Gadget	Java (HTTP/SIP Servlet)
Architecture	Server-based	Hybrid-based
Protocol	Wave (XMPP extension)	SIP

From a rather implementation point of view, the XMPP stack in the Google Wave resides at the server, and its APIs are written for third-parties to help them develop converged applications [42]. The Google team has separated the signaling (HTTP and XMPP) in a bid to maintain the current Web architecture. Hence, it could be referred to as a server-based architectural framework for service creation. In our work, the SIP stack is integrated into a browser to provide similar services. As we demonstrated, the integration of a SIP stack into the browser does not impede its performance (see Table 2), thereby making our work a viable approach to create converged services. In particular, we provide a hybrid-based architectural framework in which services are provided by both the client, using our API (TransferHTTP extension), and the proxy, using the CAS component to prevent the abuse of the services offered by the client. Hence, proxies can participate to session control so to facilitate interactions between the browsers. While the proxy services can be developed using the Mobicents SIP Servlets and JAIN SLEE APIs, the client services can be developed using our TransferHTTP extension API.

In summary, irrespective of the technologies or programming languages used in the Google Wave and TransferHTTP, the difference between them is that the Google Wave only has a stack (an XMPP stack) in its server, thereby making it a server-based architectural framework for service creation. TransferHTTP, instead, has a stack (a SIP stack) in its both client and proxy,

thereby making it a more powerful hybrid-based architectural framework for service creation.

7 Conclusion

We have shown the HTTP session mobility services available in our browser extension and proxy server. These converged services mix HTTP and SIP protocols to enhance Web 2.0 user experience. Services in the proxy, called control services, prevent abuse of the client services. In addition, examples of novel SIP-based Web 2.0 service mashups, including also VoIP and stream media to call service are discussed. Performance results obtained from TransferHTTP and CAS showed that the services at the client and proxy are not memory intensive under both low traffic and system overload conditions. They also prove that our reference system can be provisioned without necessarily using high performance PCs.

Currently, there is no agreement on what the future Internet technology will look like. It is however rather clear that the online experience can only be improved when there are additional protocols that HTTP could interact with. In fact, although the approaches to implement the services might vary, the convergence of Internet, telecommunications, and broadcasting will enable the creation of multi-protocol applications services that were not possible before.

The obtained promising results are stimulating further research activities. On the one hand, we are designing a new TransferHTTP version able to support new emerging session protocols, such as IMS. On the other hand, we are extensively evaluating and tuning the performance of our CAS component over wide-scale emulated environments.

References

1. Mozilla Add-on. (2011). https://www.addons.mozilla.org/. Accessed March 2011.
2. Canfora, G., Di Santo, G., Venturi, G., Zimeo, E., & Zito, M. V. (2005). Proxy-based Handoff of web sessions for user mobility. In *Proceedings of International conference on mobile and ubiquitous systems: Networking and services (MobiQuitous'05)*. doi:10.1109/MOBIQUITOUS.2005.50.
3. Hsieh, M.-D., Wang, T.-P., Tsai, C.-S., & Tseng, C.-C. (2006). Stateful session handoff for mobile WWW. *Information Sciences*, Elsevier Science Press, *176*(9), 1241–1265.
4. Song, H. (2002). Browser session preservation and migration. In *Poster session of world wide web (WWW)*. ISBN:1-880672-20-0.
5. The Mobicents Open Source SLEE and SIP Server. (2011). http://www.mobicents.org/index.html. Accessed March 2011.
6. Adeyeye, M., Ventura, N., & Humphrey, D. (2009). Mapping third party call control and session handoff in SIP mobility to content sharing and session handoff in the web-browsing context. In *Proceedings of IEEE wireless communications and networking conference (WCNC)*. doi:10.1109/WCNC.2009.4917800.
7. Adeyeye, M. (2008). A SIP integrated web browser for HTTP session mobility and multimedia services. Unpublished Master's thesis, University of Cape Town, South Africa.
8. Adeyeye, M., Ventura, N., & Humphrey, D. (2009). A SIP-based web session migration service. In *Proceedings of the 5th web International conference on web information systems and technologies (WEBIST)*.
9. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., et al. (2002). SIP: Session initiation protocol. IETF RFC 3261.
10. Camarillo, G., & García-Martín, M.-A. (2005). *The 3G IP multimedia subsystem (IMS)*. West Sussex, England: Wiley.
11. Gurbani, V. K., & Sun, X.-H. (2004). Terminating telephony services on the internet. *IEEE/ACM Transactions on Networking, 12*(4), 571–581.
12. The PJSIP Stack. (2011). http://www.pjsip.org. Accessed 3 March 2011.
13. The TransferHTTP Web browser Extension. (2008). http://www.transferhttp.mozdev.org. Accessed March 2011.
14. Schonwalder, J., Fouquet, M., & Rodosek, G. D. (2009). Future Internet = Content + Services + Management. *IEEE Communications Magazine, 47*(5), 27–33.
15. Mobicents SIP Servlets Server User Guide. (2011). http://www.hudson.jboss.org/hudson/view/Mobicents/job/MobicentsBooks/lastSuccessfulBuild/artifact/sip-servlets/index.html. Accessed March 2011.
16. SIPp Test Tool. (2009). http://www.sipp.sourceforge.net. Accessed March 2011.
17. PJSIP-PERF Test Tool. (2009). http://www.pjsip.org/pjsip/docs/html/page_pjsip_perf_c.htm. Accessed March 2011.
18. Jan, R. (1999). Enhancing survivability of mobile Internet access using mobile IP with location registers. In *Proceedings of IEEE INFOCOMM*, vol. 1, pp. 3–11.
19. Snoeren, A. C., & Balakrishnan, H. (2000). An end-to-end approach to host mobility. In *Proceedings of ACM MOBICOM*. doi:10.1145/345910.345938.
20. Atiquzzaman, M., Fu, S., & Ivancic, W. (2004). TraSH-SN: A transport layer seamless handoff scheme for space networks. In *Proceedings of fourth earth science technology conference (ESTC)*. doi:10.1.1.74.1729.
21. Fu, S., & Attiquzzaman, M. (2004). SCTP: State of the art in research, products, and technical challenges. *IEEE Communications Magazine, 42*(4), 64–76.
22. Stewart, R., & Metz, C. (2001). SCTP: New transport protocol for TCP/IP. *IEEE Internet Computing, 5*(6), 64–69.
23. Di Stefano, A., & Santoro, C. (2000). NetChaser: Agent support for personal mobility. *IEEE Internet Computing, 4*(2), 74–79.
24. Roussopoulos, M., Maniatis, P., Swierk, E., Lai, K., Appenzeller, G., & Baker, M. (1999). Person-level routing in the mobile people architecture. In *Proceedings of USENIX Symposium on Internet Technologies and Systems*, vol. 1, pp 165–176.
25. Maniatis, P., Roussopoulos, M., Swierk, E., Lai, K., Appenzeller, G., Zhao, X., et al. (1999). The mobile people architecture. *ACM SIGMOBILE Mobile Computing and Communications Review, 3*(3), 36–42.

26. Liscano, R., Impey, R., Qinxin, Y., & Abu-Hakima, S. (1997). Integrating multi-modal messages across heterogeneous Networks. In *Proceedings of ENM-97, in conjunction with IEEE ICC'97*. doi:10.1109/ENM.1997.596869.
27. Rao, C. H. H., Chen, Y.-F. R., Chang, D.-F., & Chen, M.-F. (2001). iMobile: A proxy-based platform for mobile services. In *Proceedings of the first workshop on wireless mobile internet*. doi:10.1145/381472.381483.
28. Raatikainen, K. (2001). Middleware for future mobile networks. In *Proceedings of IEEE Int. Conf. on 3G wireless and beyond*. ISSN:152902592.
29. Bagrodia, R., Bhattacharyya, S., Cheng, F., Gerding, S., Glazer, G., Guy, R., et al. (2003). iMASH: Interactive mobile application session handoff. In *Proceedings of the ACM International conference on mobile systems, applications, and services (MobiSys)*. doi:10.1145/1066116.1066124.
30. Greco Polito, S., & Schulzrinne, H. (2008). SIP and 802.21 for service mobility and pro-active authentication. In *Proceedings of the Communication Networks and Services Research Conf. (CNSR)*. doi:10.1109/CNSR.2008.61.
31. Lampropoulos, G., & Passas, N. (2008). Media-independent handover for seamless service provision in heterogeneous networks. *IEEE Communication Magazine, 46*(1), 64–71.
32. Tsagkaropoulos, M., Politis, I., Kotsopoulos, S., & Dagiuklas, T. (2009). Enhanced vertical handover based on 802.21 framework for real-time video streaming. In *Proceedings of the 5th International mobile multimedia communications conference (MOBIMEDIA'09)*. doi:10.4108/ICST.MOBIMEDIA2009.7347.
33. Shacham, R., Schulzrinne, H., Thakolsri, S., & Kellerer, W. (2007). Ubiquitous device personalization and use: The next generation of IP multimedia communications. *ACM Transaction on Multimedia Computing, Communications and Applications, 3*(2), 1–20.
34. The Akogrimo Project. (2011). http://www.mobilegrids.org. Accessed March 2011.
35. Wu, X., & Schulzrinne, H. (2001). Use SIP MESSAGE method for shared web browsing. http://www3.tools.ietf.org/id/draft-wu-sipping-Webshare-00.txt. Accessed 14 Nov 2001.
36. Munkongpitakkun, W., Kamolphiwong, S., & Sae-Wong, S. (2007). Enhanced web session mobility based on SIP. In *Proceedings of the 4th Int. conference on mobile technology, applications and systems (mobility)*. doi:10.1145/1378063.1378119.
37. Mulligan, C. E. A. (2009). Open API standardization for the NGN platform. *IEEE Communications Magazine, 47*(5), 108–113.
38. Krechmer, K. (2009). Open standards: A call for change. *IEEE Communications Magazine, 47*(5), 88–94.
39. Jain, R., Bakker, J., & Anjum, F. (2005). *Programming converged networks: Call control in Java, XML, and Parlay/OSA* (1st ed.). Canada: Wiley.
40. Bhat, R. R., & Tait, D. (2001). JAVA APIs for integrated networks. In T. Jespen (Ed.), *Java in telecommunications: Solutions for next generation networks* (p. 193). New York: Wiley.
41. Sun Microsystems, JAIN SIP Release 1.1 Specification. (2006). http://www.jcp.org/en/jsr/detail?id=32. Accessed March 2011.
42. Saint-Andre, P. (2004). *Extensible messaging and presence protocol (XMPP): Core*. IETF RFC 3920.
43. Web Sharing and RichDraw. (2007). http://www.hyperstruct.net/2007/2/24/xml-sync-islands-let-the-Web-sharing-begin. Accessed March 2011.
44. The Wave Protocol. (2009). http://www.waveprotocol.org. Accessed March 2011.
45. The Google Wave Project. (2009). http://www.wave.google.org. Accessed March 2011.
46. The Google Wave Click-to-dial. (2009). http://www.googlewavedev.blogspot.com/2009/06/twiliobot-bringing-phone-conversations.html. Accessed March 2011.

Author Biographies

Michael Adeyeye is a Research Fellow at the Cape Peninsula University of Technology, South Africa and a Research Associate at the University of Cape Town, South Africa, where he earned his second Master and Ph.D. degrees. His research interests include Web and multimedia service mobility technologies and context awareness, multimodal and multi-channel access, Web 2.0 and mobile applications, and Next Generation Network (NGN) applications and services. He is a Member of the IEEE.

Neco Ventura is the Head of the Centre for Broadband Networks and the Director of the Communications Research Group in the Department of Electrical Engineering at the University of Cape Town (UCT), South Africa. His current research interests are centered on Next Generation architectures, infrastructures, specifically in QoS and mobility support across heterogeneous networks. He is a Member of IEEE.

Luca Foschini is a Research Fellow at the University of Bologna, Italy. In 2007, he received a Ph.D. degree in computer science engineering from University of Bologna. His research interests include distributed systems and solutions for pervasive computing environments, system and service management, context-aware services and adaptive multimedia, and mobile agent-based middleware solutions. He is a Member of IEEE and ACM.